A tool for creating and visualizing semantic annotations on relational tables

MAZUMDAR, Suvodeep <http://orcid.org/0000-0002-0748-7638> and ZHANG, Ziqi

A Tool for Creating and Visualizing Semantic Annotations on Relational Tables

Suvodeep Mazumdar[1] and Ziqi Zhang[2]

[1]Department of Computer Science, University of Sheffield
211 Portobello, Sheffield, UK
[2]School of Science and Technology, Nottingham Trent University
50 Shakespeare Street, Nottingham, NG1 4FQ
[1]s.mazumdar@sheffield.ac.uk, [2]ziqi.zhang@ntu.ac.uk

Abstract. Semantically annotating content from relational tables on the Web is a crucial task towards realizing the vision of the Semantic Web. However, there is a lack of open source, user-friendly tools to facilitate this. This paper describes an extension of the TableMiner$^+$ system, an open source Semantic Table Interpretation system that automatically annotates Web tables using Linked Data in an effective and efficient approach. It adds a graphical user interface to TableMiner$^+$, to facilitate the visualization and correction of automatically generated annotations. This makes TableMiner$^+$ an ideal tool for the semi-automatic creation of high-quality semantic annotations on relational tables, which facilitates the publication of Linked Data on the Web.

Keywords: Web table, Named Entity Disambiguation, Semantic Table Interpretation, table annotation, Linked Data

1 Introduction

Recovering semantics from the growing amount of tabular data on the Web is a crucial task in realizing the vision of the Semantic Web. Traditional search engines perform poorly on such data, as they ignore the semantics of tabular structures [2, 3]. Recent years have seen an increase in the research on Semantic Table Interpretation [2, 5, 3, 6], which annotates relational tables using schema and entities defined in a reference knowledge base. The process deals with three types of annotation tasks in tables. Starting with the input of a well-formed relational table, it (1) links entity mentions in content cells to named entities; (2) annotates columns with concepts if they contain entity mentions, or properties of concepts if they contain data literals; and (3) identifies the semantic relations between columns. The annotations created can enable semantic indexing and search of the data, and can be used to create Linked Open Data (LOD).

Semantic Table Interpretation systems are intrinsically difficult to implement, due to, e.g., the complexity of the inter-dependent tasks (e.g., the annotation of a cell depends on on that of the containing column and vice versa), and the use of different knowledge bases. TableMiner$^+$ [7] is such a method adopting

an incremental, bootstrapping approach that starts by creating preliminary and partial annotations of a table using 'sample' data, then using the outcome as 'seed' to guide interpretation of remaining contents. This is then followed by a message passing process that iteratively refines results on the entire table to create the final optimal annotations. It has been implemented as open-source software (as part of the STI library[1]), however, the system is lacking an intuitive user interface, which has made it difficult to be used by an average person with limited technical knowledge.

This work implements a graphical user interface specifically for TableMiner$^+$, to make it an easy-to-use tool for annotating Web tables using Linked Data, and also extend it by enabling users to visualise and correct the generated annotations and Linked Data triples. As a result, data publishers can use TableMiner$^+$ for transforming tabular data on the Web into high-quality Linked Data, or creating gold-standard for experiment purposes. The remainder of this paper is structured as follows. Section 2 briefly discusses related work; Section 3 gives an overview of TableMiner$^+$; Section 4 introduces the improvement carried out in this work; Section 5 concludes this paper.

2 Related Work

Recent years have seen an increasing number of work on Semantic Table Interpretation. Venetis et al. [4] annotate columns in a table with semantic concepts and identify relations between the subject column (typically containing entities that the table is about) and other columns using a database mined with regular lexico-syntactic patterns such as the Hearst patterns [1]. The database records co-occurrence statistics for each pair of values extracted by such patterns. A maximum likelihood inference model is used to predict the best concepts and relations from candidates using these statistics. Limaye et al. [2] uses a joint inference model, i.e., factor graph to model a table and the interdependencies between its components. Table components are modeled as variables represented as nodes on the graph; then the interdependencies among variables are modeled by factors. The task of inference amounts to searching for an assignment of values to the variables that maximizes the joint probability. Mulwad et al. [3] also uses joint inference with semantic message passing. TableMiner [6] and TableMiner$^+$[7] adopt a bootstrapping approach starting by creating preliminary annotations of a table using automatically selected 'sample' data in the table, followed by a message passing process that iteratively refines the preliminary annotations to create the final optimal results. These methods differ in terms of the inference models, features and background knowledge bases used. As discussed before, existing tools remains difficult to use due to the lack of a user friendly interface.

[1] https://github.com/ziqizhang/sti

3 Overview of TableMiner$^+$

Figure 1 shows a high-level view of the components and workflow of TableMiner$^+$. We refer readers to Zhang [7] for details of the methodology. The system can be divided into three major components. Firstly, it detects a 'subject column' (**SUBJECT COLUMN DETECTION**), which is the one in the table containing named entities that are subjects of each rows. TableMiner$^+$ assumes other columns in a relational table are data describing the subjects. It then identifies other columns that also contain named entities (NE-columns), and performs column classification (assigning a URI from a knowledge base to the column) and cell disambiguation (assigning a URI from a knowledge base to each cell) on these as well as the subject columns. Working with each NE-column at a time, these are further divided into two processes. In the **LEARNING** phase, the system attempts to use a subset (*Sample Ranking*) of rows from the NE-column to infer a concept URI for the column (*Preliminary Col. Classify. with I-Inf*). The idea is that, usually for human-beings, we only need to see some (and rarely do we need to see all) data in a column in order to classify them. However, it is likely that our understanding could be biased because of this 'partial' view. And therefore, we call these results 'preliminary', which will be optimized later. The LEARNING phase also uses preliminary column annotations as input to guide *Preliminary Cell Disambiguation*. In this part of the process, the assigned concept URI for the column determines the candidate named entities for each row in that column. Next, the preliminary annotations for a column and its content cells are optimized in the **UPDATE** phase. In this phase, the system attempts to ensure annotations on different NE-columns are consistent, e.g., they belong to the same domain (*Compute Domain Representation*). The computation can alter the preliminary annotations in some columns or content cells, which then causes a chain of alterations due to the interdependency of the tasks. A semantic message passing algorithm is implemented to control such update process until convergence. With the column and cell annotations finalized, TableMiner$^+$ moves on to infer relations between the subject column and other columns (**RELATION ENUMERATION + LITERAL COLUMN ANNOTATION**). In simple terms, the relation between a subject column and another column is selected based on the relations derived on each row between the pairs of subject entity and data in the other column.

4 Description of the TableMiner$^+$ Application Interface

In this section, we describe the TableMiner$^+$ user interface and the use of the tool through this interface. We use the implementation distributed as part of the STI library as basis for this work. The STI library provides an implementation of the system introduced in Zhang [7], and a few baseline systems. The library is implemented in Java, and uses DBpedia as the knowledge base. Currently, a Web-based interface consisting of two components are implemented: one that lets users to define, configure and start a table annotation task; and the other that

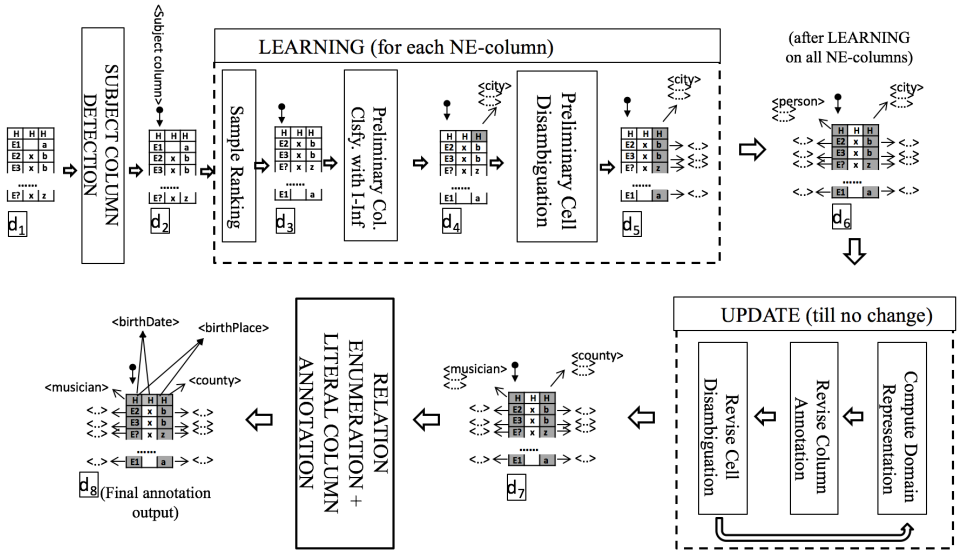

Fig. 1. Overall component and workflow of TableMiner$^+$

lets users to visualise and correct annotation results. In both cases, interaction is achieved via a Web browser[2].

4.1 Starting a table annotation process

The interface for starting a Semantic Table Annotation task is illustrated in Figure 2[3]. Users firstly enter the URL of a webpage containing relational tables that are to be annotated. Upon entering the details, the user is shown a preview of the page along with a highlighted list of tables potentially containing relational data. The users then select the tables they wish to annotate. They can also configure the system to alter settings such as feature weights and knowledge base query constraints. The users may provide an email address to subscribe for an automatic alert when the annotation task completes. When the users are satisfied with the configuration and the input, they can click the button to start the task, which will create annotations in JSON format. These will be interpreted and displayed using the visualisation component described below.

4.2 Visualisation and correction of annotations

The JSON files are then passed onto the visualisation component, which consists of two interactive elements: an annotated table and a graph visualisation module.

[2] However, it is not recommended to deploy TableMiner$^+$ as a Web-service as it does not support concurrent access typically found in multi-user environment.

[3] Follow https://github.com/ziqizhang/sti/tree/master/ui for a demo and on how to use

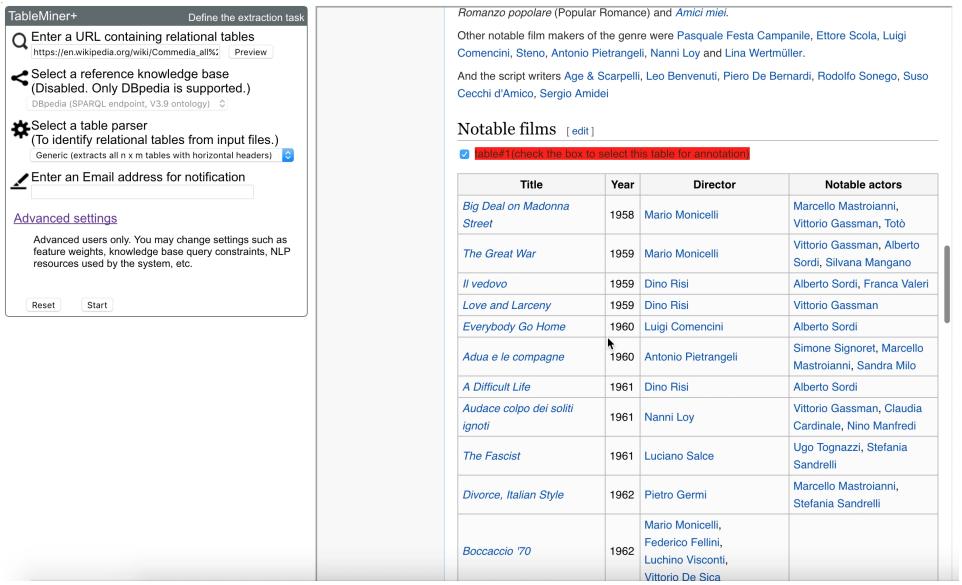

Fig. 2. User input interface, where the user enters source URL containing the respective relational tables. The preview button provides an idea of the table that will be annotated, and proceeding with start initiates the extraction process. An email address can be provided if the user prefers not to wait and be notified once the extraction process has completed.

The annotated table is the first point of interaction with the user, and presents the original table, annotated with the entities, concepts and relations identified by TableMiner$^+$. The first step for the UI is to investigate the header cells of the table - TableMiner$^+$ creates a set of candidate concepts that best describe the header and the data in the column. Each associated concept has a score indicating the system's confidence. This set of candidate concepts is presented as a dropdown with the scores (Figure 3 section B). Users can select any of the concepts to indicate a more appropriate annotation by clicking on the respective concept. Concepts are further encoded on the basis of scores (the highest scoring concepts are indicated in green, while the lowest in red), which provides an indication of the confidence in content cell annotations can be visualised in the same way.

As can also be seen from the figure, some entities have already been recognized, while some haven't. In such cases the user can provide a URI that is appropriate for any missing annotation, this can be done by clicking the relevant cell, which will provide a prompt for a text input (Figure 4). Further SPARQL queries can also be triggered to the respective endpoints (based on the user customisations) that can identify any missing annotations.

While tables can provide a clean annotated replication of the original source document, with the added ability for users to provide their annotations and

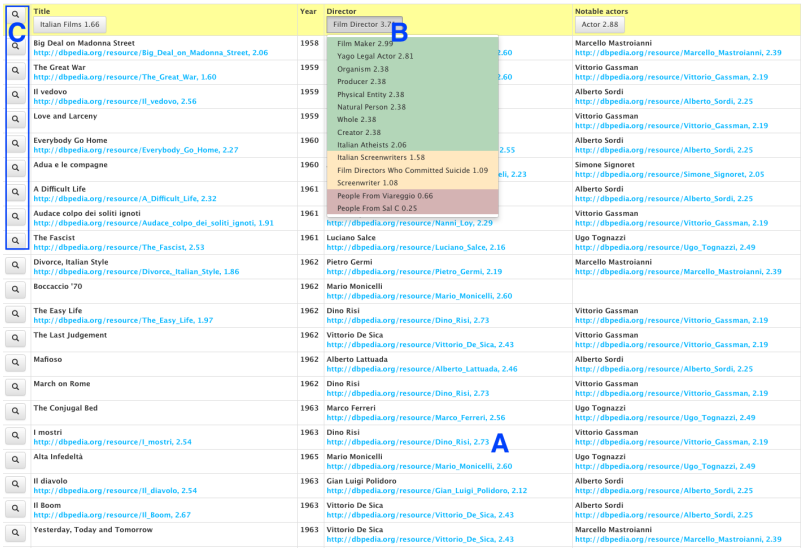

Fig. 3. TableMiner UI Annotated Table for the page available at `https://en.wikipedia.org/wiki/Commedia_all\%27italiana`

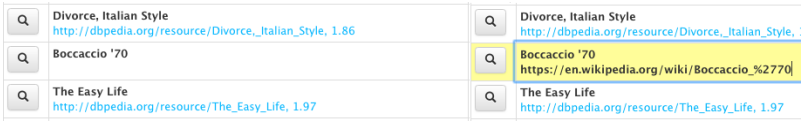

Fig. 4. Editing cell contents to add annotations - for missing/wrong annotations (as seen in the figure left), clicking on the relevant cell will enable editing (right). All changes made on the tables are stored until finally pushed to the backend database

correct any mistakes they can observe, a further need may arise for greater customisation and control of annotations. This provides users with means to visualise (and annotate) possible relations among table columns, in addition to visualising possible candidate annotations. The next aspect of the UI is the graph visualisation, which is invoked from the 'inspect' button on the first cell of each table row (Figure 3, Section C). As an example, the header and it's relevant candidate concepts have been plotted as a graph in Figure 5.

Header cells are shown as nodes labelled with the header columns (0-3), while the candidate classes are shown as nodes, linked with header elements. The most relevant class is shown with a strong link, while the others are presented as dashed lines. Clicking on an individual node makes all other nodes more transparent, and hence keeps the current node and link in focus. Right-clicking the dashed ones annotate the relevant header cell with the respective concept, which will then confirm the change with a strong link (here, a straight thick line). Header cells are also linked with each other with dashed lines, which is

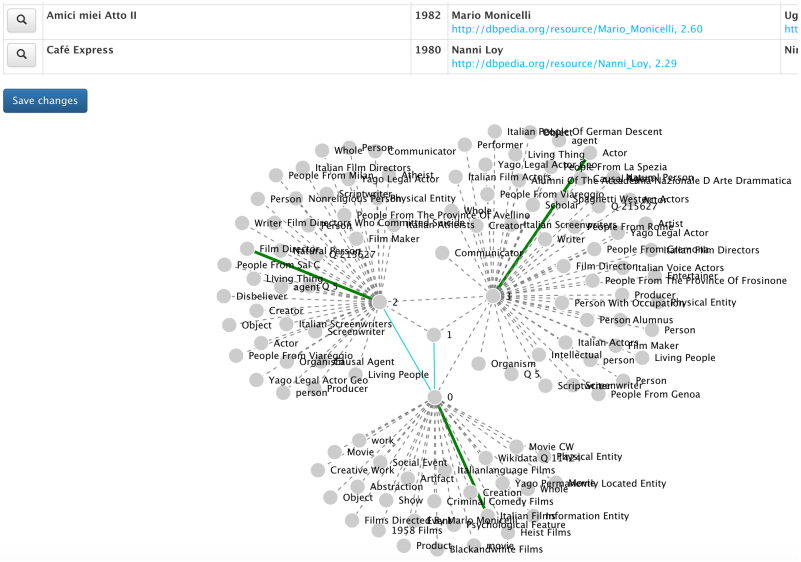

Fig. 5. Visualising table rows as a node-link graph

interpreted as only an indicative relation. However, if TableMiner$^+$ creates any relations between the columns, it is reflected as straight lines as can be seen in Figure 5.

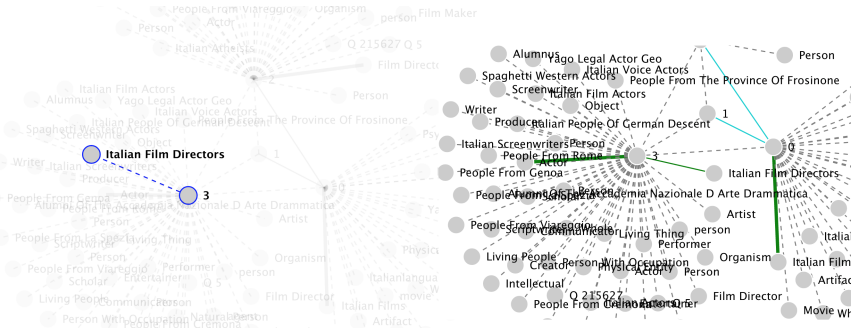

Fig. 6. Clicking on nodes focusses the view on the clicked node and immediate connections to other nodes. Right clicking a node creates a stronger connection (right, a new link is established with 'Italian Film Directors') which is interpreted as another annotation for the column in the original table

Described Scenario In the example shown so far, the source URL (`https://en.wikipedia.org/wiki/Commedia_all\%27italiana`) describes movies re-

leased that belong to an Italian film genre. The extraction process in TableMiner$^+$ annotated several cells of the table selected by the user in Figure 2 (Notable films), however several films could not be identified. This is made evident when the user visualises the annotated table (Figure 3). Missing cells can then be manually annotated by adding URLs if the user can provide any unidentified ones (Figure 4). For example, the user observes the cell 'Boccaccio ́70' could not be identified and hence chose to manually add the resource URL. Further inspecting the different concepts, users can click on the 'inspect' button to visualise the concepts on a node-link graph (Figure 5). While interacting with the graph, the user notes that since the original webpage discussed Italian movie genre, the 'Italian Film Directors' concept would be appropriate to describe the first column in the table. Hence the user can right-click on the concept to add a new annotation for the column. Each row in the table can be visualised as a graph, and hence the user can introduce row-specific annotations as well. Finally, when all annotations are completed, the user can click on 'Save changes' to submit all annotations to TableMiner$^+$.

5 Conclusion

This paper introduced a graphical user interface for TableMiner$^+$ to facilitate the semi-automatic creation of high quality Linked Data and annotations on Web tables. Future work will extend the system to support, e.g., different knowledge bases, other algorithms, fine-grained task definition that enable batch processing and zoning on tables (e.g., specific columns). Furthermore, we will also explore different visualisations and mechanisms for users to introduce new annotations, visualising relevant sections of ontologies while exploring table annotations. We also have a series of user evaluations planned to understand how users can make use of the user interface.

Acknowledgement This work is funded by the EU FP7 WeSenseIt (grant agreement 308429)[4] and EU Horizon 2020 Seta (grant agreement 688082)[5] projects. We also thank the ADEQUATe[6] project team under the lead of Dr Tomas Knap for contributing valuable design ideas.

References

1. Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 2*, COLING '92, pages 539–545, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[4] http://wesenseit.eu/
[5] http://setamobility.eu/
[6] http://www.adequate.at

2. Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment*, 3(1-2):1338–1347, 2010.
3. Varish Mulwad, Tim Finin, and Anupam Joshi. Semantic message passing for generating linked data from tables. In *International Semantic Web Conference (1)*, Lecture Notes in Computer Science, pages 363–378. Springer, 2013.
4. Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *Proceedings of VLDB Endowment*, 4(9):528–538, June 2011.
5. Jingjing Wang, Haixun Wang, Zhongyuan Wang, and Kenny Q. Zhu. Understanding tables on the web. In *Proceedings of the 31st international conference on Conceptual Modeling*, ER'12, pages 141–155, Berlin, Heidelberg, 2012. Springer-Verlag.
6. Ziqi Zhang. Towards effective and efficient semantic table interpretation. In *Proceedings of the 13th International Semantic Web Conference*, pages 487–502, 2014.
7. Ziqi Zhang. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web Journal*, Accepted. Tracking: 1339-2551, 2016.